In [1]:

```
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen

Upload widget is only available when the cell has been executed in the current browser session.
Please rerun this cell to enable.

In [0]:

```
import os
```

In [6]:

```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
!ls ~/.kaggle
```

```
kaggle.json
```

In [3]:

```
!ls -l ~/.kaggle
!cat ~/.kaggle/kaggle.json
```

```
total 4
-rw------- 1 root root 62 Apr 29 00:20 kaggle.json
{"username":"alkong","key":"83ed69f445757583db92e9604d9a915d"}
```

In [0]:

```
!pip install -q kaggle
!pip install -q kaggle-cli
```

In [5]:

```
!kaggle datasets download paultimothymooney/breast-histopathology-images
```

```
Downloading breast-histopathology-images.zip to /content
 99% 1.47G/1.49G [00:10<00:00, 128MB/s]
100% 1.49G/1.49G [00:10<00:00, 148MB/s]
```

In [6]:

```
!mkdir data
!mv breast-histopathology-images.zip data/
```

```
mkdir: cannot create directory ‘data’: File exists
```

In [7]:

```
!unzip data/breast-histopathology-images.zip
```

```
Archive:  data/breast-histopathology-images.zip
  inflating: IDC_regular_ps50_idx5.zip
```

In [ ]:

```
import zipfile

zip_file = zipfile.ZipFile('IDC_regular_ps50_idx5.zip', 'r')
zip_file.extractall('/content')
zip_file.close()
```

In [0]:

```
!rm "kaggle (1).json" "kaggle (2).json"
```

In [0]:

```
!rm IDC_regular_ps50_idx5.zip
```

In [0]:

```
import cv2
import os

files = os.listdir('/content')
new_files = [f for f in files if f != '.config' and f != 'adc.json' and f != 'kaggle.js
on' and f != 'sample_data']
class0 = []
class1 = []
for i in range(0,20):
  f1 = new_files[i]
  for classes in os.listdir('/content/'+ f1):
     for img in os.listdir('/content/'+ f1+ '/' + classes):
        if classes == '0':
          class0.append(cv2.imread('/content/' + f1+ '/' + classes + '/' + img))
        else:
          class1.append(cv2.imread('/content/' + f1+ '/' + classes + '/' + img))
```

In [15]:

```
print(class0[0].shape)
print(class1[0].shape)
```

```
(50, 50, 3)
(50, 50, 3)
```

In [16]:

```
import numpy as np

class0_clean = []
class1_clean = []

for img in class0:
  if img is None or img.shape[0] != 50 or img.shape[1] != 50:
    continue
  class0_clean.append(img)

for img in class1:
  if img is None or img.shape[0] != 50 or img.shape[1] != 50:
    continue
  class1_clean.append(img)

class0_data = np.array(class0_clean)
class1_data = np.array(class1_clean)

print(class0_data.shape)
print(class1_data.shape)


train_array = np.append(class0_data, class1_data,axis=0)
```

```
(18631, 50, 50, 3)
(4889, 50, 50, 3)
```

In [0]:

```
import pandas as pd
from keras.datasets import mnist
import keras
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout
import numpy as np
from keras.layers.normalization import BatchNormalization
```

In [18]:

```
train_array.shape
```

Out[18]:

```
(23520, 50, 50, 3)
```

In [0]:

```
y_1 = np.ones(4889)
y_0 = np.zeros(18631)
y = np.append(y_0,y_1,axis=0)
```

In [0]:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(train_array, y, test_size=0.2, rand
om_state=42)
```

Q 3.1

CNN with dropout and batch normalization

In [0]:

```
from keras.layers import Conv2D, MaxPooling2D, Flatten

def run_cnn(layers,dropout,units,X_input, y_input, epochs):
  model = Sequential()
  model.add(Conv2D(256, (3,3), activation='relu', input_shape=(X_train.shape[1:])))
  model.add(Flatten())

  for i in range(0,layers):
    model.add(Dense(units, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(dropout))

  model.add(Dense(1, activation='sigmoid'))
  model.compile(loss='binary_crossentropy',
            optimizer='adam',
            metrics=['accuracy'])
  history_callback = model.fit(X_input, y_input, batch_size=50, epochs=epochs, verbose=
1, validation_split=0.2)
  return pd.DataFrame(history_callback.history), model
```

In [32]:

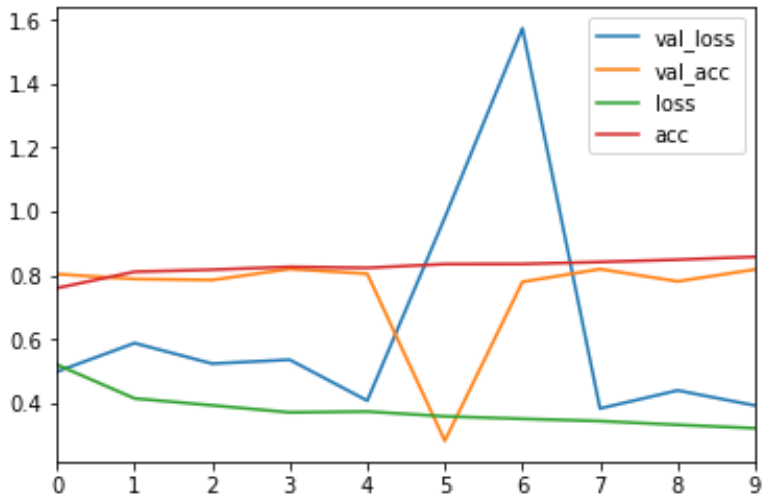
```
results, model = run_cnn(3,0.3,64, X_train, y_train, 10)
print(results)
results.plot()
```

```
Train on 15052 samples, validate on 3764 samples
Epoch 1/10
15052/15052 [==============================] - 18s 1ms/step - loss: 0.5199
- acc: 0.7599 - val_loss: 0.4978 - val_acc: 0.8037
Epoch 2/10
15052/15052 [==============================] - 16s 1ms/step - loss: 0.4140
- acc: 0.8110 - val_loss: 0.5874 - val_acc: 0.7883
Epoch 3/10
15052/15052 [==============================] - 16s 1ms/step - loss: 0.3923
- acc: 0.8174 - val_loss: 0.5231 - val_acc: 0.7851
Epoch 4/10
15052/15052 [==============================] - 16s 1ms/step - loss: 0.3703
- acc: 0.8264 - val_loss: 0.5353 - val_acc: 0.8196
Epoch 5/10
15052/15052 [==============================] - 16s 1ms/step - loss: 0.3729
- acc: 0.8232 - val_loss: 0.4070 - val_acc: 0.8047
Epoch 6/10
15052/15052 [==============================] - 15s 1ms/step - loss: 0.3579
- acc: 0.8350 - val_loss: 0.9808 - val_acc: 0.2811
Epoch 7/10
15052/15052 [==============================] - 16s 1ms/step - loss: 0.3502
- acc: 0.8354 - val_loss: 1.5741 - val_acc: 0.7792
Epoch 8/10
15052/15052 [==============================] - 16s 1ms/step - loss: 0.3430
- acc: 0.8415 - val_loss: 0.3825 - val_acc: 0.8193
Epoch 9/10
15052/15052 [==============================] - 16s 1ms/step - loss: 0.3313
- acc: 0.8489 - val_loss: 0.4392 - val_acc: 0.7808
Epoch 10/10
15052/15052 [==============================] - 16s 1ms/step - loss: 0.3206
- acc: 0.8579 - val_loss: 0.3919 - val_acc: 0.8183
```

	val_loss	val_acc	loss	acc
0	0.497773	0.803666	0.519858	0.759899
1	0.587438	0.788257	0.414006	0.810989
2	0.523064	0.785069	0.392305	0.817433
3	0.535291	0.819607	0.370257	0.826402
4	0.406980	0.804729	0.372879	0.823213
5	0.980797	0.281084	0.357863	0.835039
6	1.574071	0.779224	0.350172	0.835371
7	0.382456	0.819341	0.343046	0.841549
8	0.439168	0.780818	0.331328	0.848924
9	0.391884	0.818278	0.320608	0.857893

Out[32]:

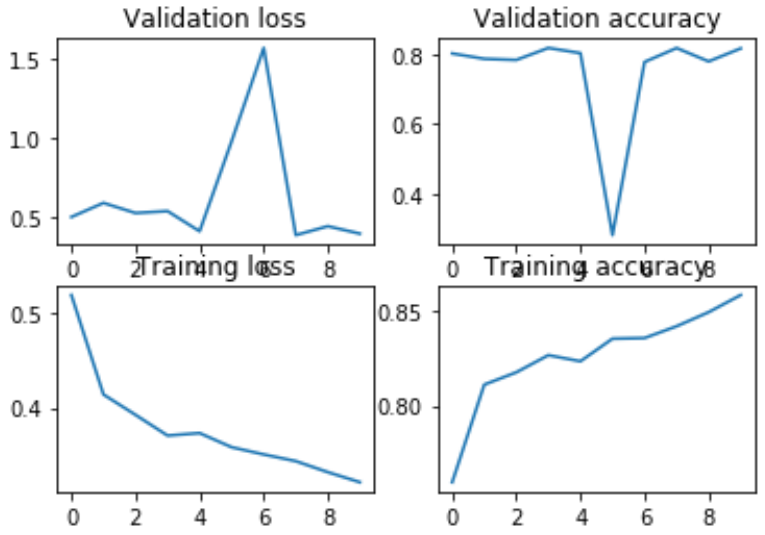
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe2143afba8>
```



In [33]:

```
import matplotlib.pyplot as plt

fig, axes = plt.subplots(2, 2)
_ = axes[0,0].plot(pd.Series(results.index),results['val_loss'])
_ = axes[0,0].set_title('Validation loss')
_ = axes[0,1].plot(pd.Series(results.index),results['val_acc'])
_ = axes[0,1].set_title('Validation accuracy')
_ = axes[1,0].plot(pd.Series(results.index),results['loss'])
_ = axes[1,0].set_title('Training loss')
_ = axes[1,1].plot(pd.Series(results.index),results['acc'])
_ = axes[1,1].set_title('Training accuracy')
```



In [34]:

```
score = model.evaluate(X_test, y_test, verbose=0)
print("Test loss: {:.3f}".format(score[0]))
print("Test Accuracy: {:.3f}".format(score[1]))
```

```
Test loss: 0.400
Test Accuracy: 0.809
```

Q 3.2 Rotation

In [35]:

```
X_train[0].shape
```

Out[35]:

```
(50, 50, 3)
```

In [0]:

```
import cv2

rows,cols = X_train[0].shape[0], X_train[0].shape[1]

M = cv2.getRotationMatrix2D((50/2,50/2),90,1)
imgs_rotated = []

for img in X_train:
  imgs_rotated.append(cv2.warpAffine(img,M,(50,50)))

X_train_rotated = np.array(imgs_rotated)
```

In [39]:

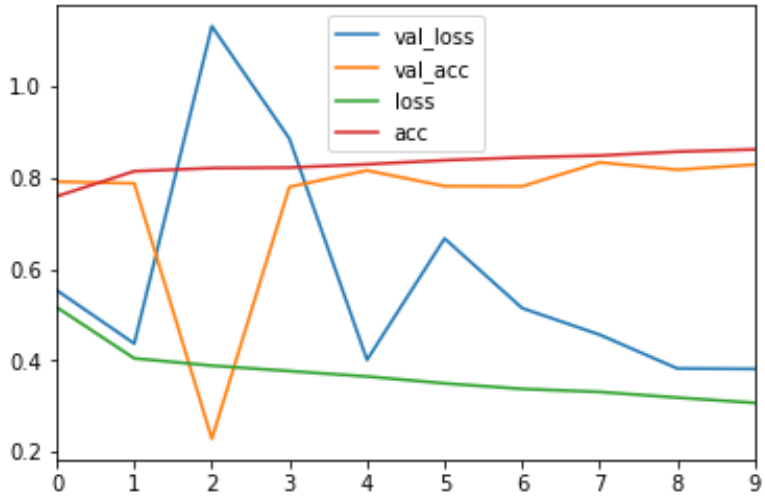
```
# using baseline
results, model_rotated = run_cnn(3,0.3,64, X_train_rotated, y_train,10)
print(results)
results.plot()
```

```
Train on 15052 samples, validate on 3764 samples
Epoch 1/10
15052/15052 [==============================] - 18s 1ms/step - loss: 0.5163
- acc: 0.7588 - val_loss: 0.5535 - val_acc: 0.7909
Epoch 2/10
15052/15052 [==============================] - 15s 983us/step - loss: 0.40
45 - acc: 0.8137 - val_loss: 0.4369 - val_acc: 0.7872
Epoch 3/10
15052/15052 [==============================] - 15s 984us/step - loss: 0.38
85 - acc: 0.8206 - val_loss: 1.1308 - val_acc: 0.2285
Epoch 4/10
15052/15052 [==============================] - 15s 982us/step - loss: 0.37
67 - acc: 0.8218 - val_loss: 0.8850 - val_acc: 0.7795
Epoch 5/10
15052/15052 [==============================] - 15s 988us/step - loss: 0.36
48 - acc: 0.8289 - val_loss: 0.4012 - val_acc: 0.8154
Epoch 6/10
15052/15052 [==============================] - 15s 998us/step - loss: 0.34
98 - acc: 0.8377 - val_loss: 0.6666 - val_acc: 0.7811
Epoch 7/10
15052/15052 [==============================] - 15s 986us/step - loss: 0.33
77 - acc: 0.8439 - val_loss: 0.5143 - val_acc: 0.7806
Epoch 8/10
15052/15052 [==============================] - 15s 987us/step - loss: 0.33
11 - acc: 0.8480 - val_loss: 0.4562 - val_acc: 0.8332
Epoch 9/10
15052/15052 [==============================] - 15s 990us/step - loss: 0.31
87 - acc: 0.8565 - val_loss: 0.3824 - val_acc: 0.8170
Epoch 10/10
15052/15052 [==============================] - 15s 989us/step - loss: 0.30
72 - acc: 0.8614 - val_loss: 0.3814 - val_acc: 0.8284
```

	val_loss	val_acc	loss	acc
0	0.553477	0.790914	0.516343	0.758836
1	0.436862	0.787194	0.404503	0.813712
2	1.130781	0.228480	0.388542	0.820555
3	0.885040	0.779490	0.376739	0.821751
4	0.401174	0.815356	0.364846	0.828926
5	0.666606	0.781084	0.349816	0.837696
6	0.514343	0.780553	0.337701	0.843875
7	0.456244	0.833156	0.331077	0.847994
8	0.382396	0.816950	0.318698	0.856497
9	0.381437	0.828374	0.307157	0.861414

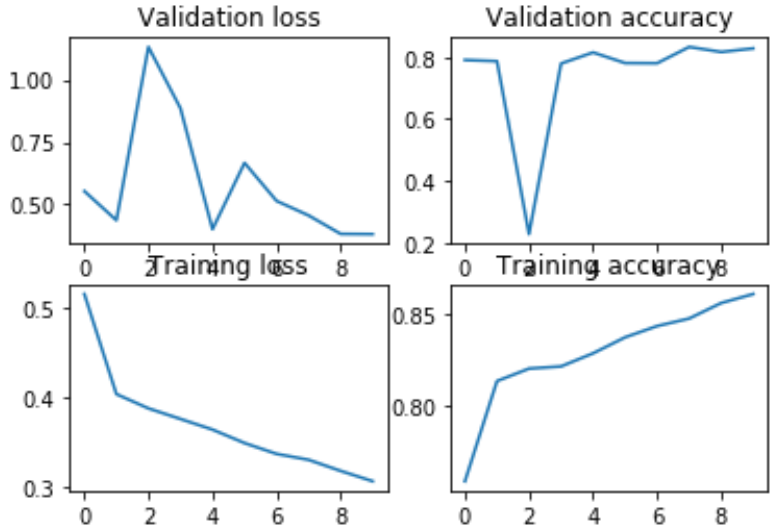
Out[39]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe211f474a8>
```



In [40]:

```
fig, axes = plt.subplots(2, 2)
_ = axes[0,0].plot(pd.Series(results.index),results['val_loss'])
_ = axes[0,0].set_title('Validation loss')
_ = axes[0,1].plot(pd.Series(results.index),results['val_acc'])
_ = axes[0,1].set_title('Validation accuracy')
_ = axes[1,0].plot(pd.Series(results.index),results['loss'])
_ = axes[1,0].set_title('Training loss')
_ = axes[1,1].plot(pd.Series(results.index),results['acc'])
_ = axes[1,1].set_title('Training accuracy')
```



In [41]:

```
score = model_rotated.evaluate(X_test, y_test, verbose=0)
print("Test loss: {:.3f}".format(score[0]))
print("Test Accuracy: {:.3f}".format(score[1]))
```

```
Test loss: 0.376
Test Accuracy: 0.820
```

Q 3.2 Vertical flip - mirroring

In [42]:

```
imgs_mirrored = []

for img in X_train:
  imgs_mirrored.append(cv2.flip(img,1))

X_train_mirrored = np.array(imgs_mirrored)
print(X_train_mirrored.shape)
```

```
(18816, 50, 50, 3)
```

In [48]:

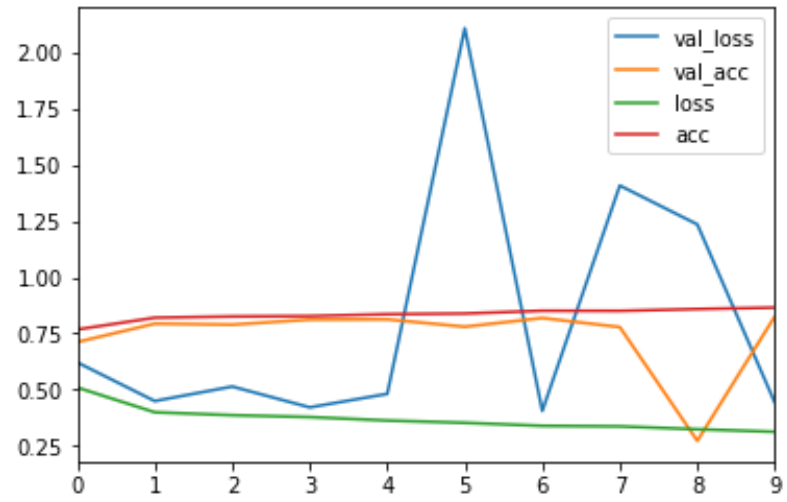
```
# using original baseline
results, model_flip = run_cnn(3,0.2,64, X_train_mirrored, y_train,10)
print(results)
results.plot()
```

```
Train on 15052 samples, validate on 3764 samples
Epoch 1/10
15052/15052 [==============================] - 19s 1ms/step - loss: 0.5089
- acc: 0.7673 - val_loss: 0.6209 - val_acc: 0.7112
Epoch 2/10
15052/15052 [==============================] - 15s 1ms/step - loss: 0.3984
- acc: 0.8199 - val_loss: 0.4485 - val_acc: 0.7933
Epoch 3/10
15052/15052 [==============================] - 15s 1ms/step - loss: 0.3854
- acc: 0.8254 - val_loss: 0.5130 - val_acc: 0.7891
Epoch 4/10
15052/15052 [==============================] - 15s 1ms/step - loss: 0.3770
- acc: 0.8271 - val_loss: 0.4203 - val_acc: 0.8114
Epoch 5/10
15052/15052 [==============================] - 15s 995us/step - loss: 0.36
16 - acc: 0.8356 - val_loss: 0.4803 - val_acc: 0.8124
Epoch 6/10
15052/15052 [==============================] - 15s 983us/step - loss: 0.35
14 - acc: 0.8384 - val_loss: 2.1101 - val_acc: 0.7798
Epoch 7/10
15052/15052 [==============================] - 15s 1ms/step - loss: 0.3380
- acc: 0.8509 - val_loss: 0.4045 - val_acc: 0.8185
Epoch 8/10
15052/15052 [==============================] - 16s 1ms/step - loss: 0.3356
- acc: 0.8503 - val_loss: 1.4090 - val_acc: 0.7782
Epoch 9/10
15052/15052 [==============================] - 16s 1ms/step - loss: 0.3227
- acc: 0.8584 - val_loss: 1.2358 - val_acc: 0.2710
Epoch 10/10
15052/15052 [==============================] - 16s 1ms/step - loss: 0.3115
- acc: 0.8650 - val_loss: 0.4412 - val_acc: 0.8262
```

	val_loss	val_acc	loss	acc
0	0.620902	0.711211	0.508889	0.767340
1	0.448455	0.793305	0.398364	0.819891
2	0.513030	0.789054	0.385371	0.825405
3	0.420253	0.811371	0.377014	0.827133
4	0.480342	0.812434	0.361630	0.835636
5	2.110124	0.779756	0.351436	0.838360
6	0.404489	0.818544	0.337994	0.850850
7	1.408978	0.778162	0.335559	0.850319
8	1.235752	0.270988	0.322667	0.858424
9	0.441226	0.826249	0.311502	0.865001

Out[48]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe10caafe10>
```



In [49]:

```
score = model_flip.evaluate(X_test, y_test, verbose=0)
print("Test loss: {:.3f}".format(score[0]))
print("Test Accuracy: {:.3f}".format(score[1]))
```

```
Test loss: 0.450
Test Accuracy: 0.812
```

Q3.2 Image Translation

In [50]:

```
imgs_translated = []

for img in X_train:
  imgs_translated.append(cv2.flip(img,1))

X_train_translated = np.array(imgs_translated)
print(X_train_translated.shape)
```

```
(18816, 50, 50, 3)
```

In [51]:

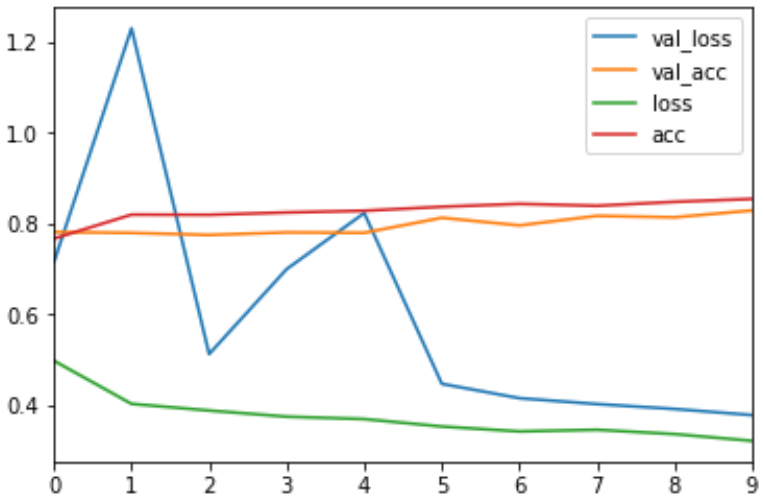
```
# baseline
results, model_translated = run_cnn(3,0.2,64, X_train_translated, y_train,10)
print(results)
results.plot()
```

```
Train on 15052 samples, validate on 3764 samples
Epoch 1/10
15052/15052 [==============================] - 20s 1ms/step - loss: 0.4979
- acc: 0.7667 - val_loss: 0.7151 - val_acc: 0.7811
Epoch 2/10
15052/15052 [==============================] - 15s 979us/step - loss: 0.40
32 - acc: 0.8194 - val_loss: 1.2293 - val_acc: 0.7792
Epoch 3/10
15052/15052 [==============================] - 15s 983us/step - loss: 0.38
85 - acc: 0.8190 - val_loss: 0.5127 - val_acc: 0.7747
Epoch 4/10
15052/15052 [==============================] - 15s 980us/step - loss: 0.37
50 - acc: 0.8241 - val_loss: 0.6996 - val_acc: 0.7803
Epoch 5/10
15052/15052 [==============================] - 15s 977us/step - loss: 0.36
98 - acc: 0.8279 - val_loss: 0.8229 - val_acc: 0.7795
Epoch 6/10
15052/15052 [==============================] - 15s 987us/step - loss: 0.35
33 - acc: 0.8367 - val_loss: 0.4472 - val_acc: 0.8124
Epoch 7/10
15052/15052 [==============================] - 15s 988us/step - loss: 0.34
26 - acc: 0.8431 - val_loss: 0.4157 - val_acc: 0.7957
Epoch 8/10
15052/15052 [==============================] - 15s 980us/step - loss: 0.34
61 - acc: 0.8388 - val_loss: 0.4027 - val_acc: 0.8167
Epoch 9/10
15052/15052 [==============================] - 15s 979us/step - loss: 0.33
67 - acc: 0.8475 - val_loss: 0.3919 - val_acc: 0.8135
Epoch 10/10
15052/15052 [==============================] - 15s 981us/step - loss: 0.32
18 - acc: 0.8539 - val_loss: 0.3785 - val_acc: 0.8286
```

	val_loss	val_acc	loss	acc
0	0.715104	0.781084	0.497948	0.766676
1	1.229331	0.779224	0.403183	0.819360
2	0.512688	0.774708	0.388479	0.818961
3	0.699586	0.780287	0.375029	0.824143
4	0.822932	0.779490	0.369769	0.827930
5	0.447236	0.812434	0.353346	0.836699
6	0.415655	0.795696	0.342578	0.843077
7	0.402737	0.816684	0.346149	0.838825
8	0.391898	0.813496	0.336689	0.847529
9	0.378496	0.828640	0.321820	0.853906

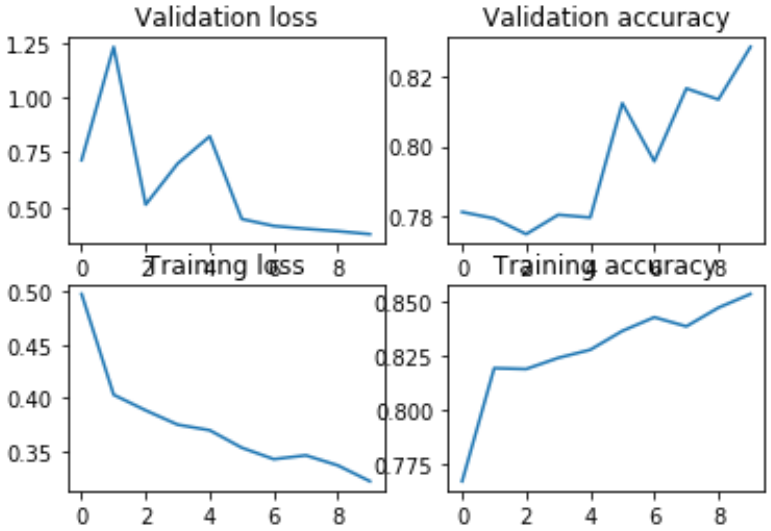
Out[51]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe10b86f4e0>
```



In [52]:

```
fig, axes = plt.subplots(2, 2)
_ = axes[0,0].plot(pd.Series(results.index),results['val_loss'])
_ = axes[0,0].set_title('Validation loss')
_ = axes[0,1].plot(pd.Series(results.index),results['val_acc'])
_ = axes[0,1].set_title('Validation accuracy')
_ = axes[1,0].plot(pd.Series(results.index),results['loss'])
_ = axes[1,0].set_title('Training loss')
_ = axes[1,1].plot(pd.Series(results.index),results['acc'])
_ = axes[1,1].set_title('Training accuracy')
```



In [53]:

```
score = model_translated.evaluate(X_test, y_test, verbose=0)
print("Test loss: {:.3f}".format(score[0]))
print("Test Accuracy: {:.3f}".format(score[1]))
```

```
Test loss: 0.387
Test Accuracy: 0.822
```

Q 3.3 Deep model with Residual Connections

In [0]:

```
from keras.layers import Input, Dense
from keras.models import Model


def run_cnn_deep(layers,dropout,units,X_input, y_input, epochs):

  inputs = Input(shape=(50,50,3))
  x = Conv2D(256, (3,3), activation='relu', input_shape=(X_train.shape[1:]))(inputs)
  # x = Conv2D(256, (3,3), activation='relu', input_shape=(X_train.shape[1:]))
  x = Flatten()(x)
  x = Dense(units, activation='relu')(x)
  print('About to add layers iteratively')

  for i in range(0,layers-1):
  x = Conv2D(256, (3,3), activation='relu', input_shape=(X_train.shape[1:]))(x)
    x = Flatten()(x)
    x = Dense(units, activation='relu')(x)
    # x = Dropout(dropout)(x)


  predictions = Dense(1, activation='sigmoid')(x)
  model = Model(inputs=inputs, outputs=predictions)
  print(model.summary())
  model.compile(loss='binary_crossentropy',
            optimizer='adam',
            metrics=['accuracy'])
  history_callback = model.fit(X_input, y_input, batch_size=50, epochs=epochs, verbose=
1, validation_split=0.2)
  return pd.DataFrame(history_callback.history), model
```

In [21]:

```
from keras.layers import Input, Conv2D, MaxPooling2D, Flatten,add,Dense
from keras.models import Model
num_classes = 10
inputs = Input(shape=(50, 50, 3))

conv1 = Conv2D(32, (3, 3), activation='relu',
                padding='same')(inputs)
conv2 = Conv2D(32, (3, 3), activation='relu',
                padding='same')(conv1)
maxpool1 = MaxPooling2D(pool_size=(2, 2))(conv2)
conv3 = Conv2D(32, (3, 3), activation='relu',
                padding='same')(maxpool1)
conv4 = Conv2D(32, (3, 3), activation='relu',
                padding='same')(conv3)
maxpool2 = MaxPooling2D(pool_size=(2, 2))(conv4)

conv5 = Conv2D(32, (3, 3), activation='relu',
                padding='same')(maxpool2)
conv6 = Conv2D(32, (3, 3), activation='relu',
                padding='same')(conv5)
maxpool3 = MaxPooling2D(pool_size=(2, 2))(conv6)
conv7 = Conv2D(32, (3, 3), activation='relu',
                padding='same')(maxpool3)
conv8 = Conv2D(32, (3, 3), activation='relu',
                padding='same')(conv7)


skip2 = add([maxpool3, conv8])
maxpool4 = MaxPooling2D(pool_size=(2, 2))(skip2)
flat = Flatten()(maxpool4)
dense = Dense(10, activation='relu')(flat)
predictions = Dense(1, activation='sigmoid')(dense)
model = Model(inputs=inputs, outputs=predictions)

model.compile(loss='binary_crossentropy',
            optimizer='adam',
            metrics=['accuracy'])
history_callback = model.fit(X_train, y_train, batch_size=50, epochs=10, verbose=1, val
idation_split=0.2)
```

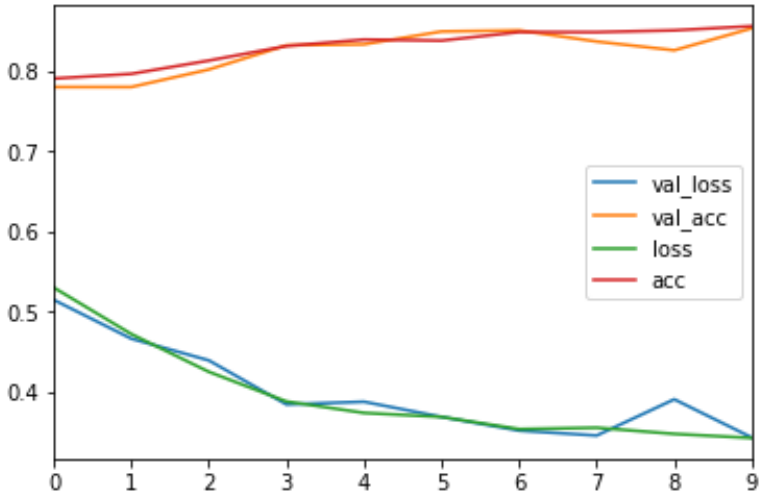
```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/
python/ops/math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_op
s) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Train on 15052 samples, validate on 3764 samples
Epoch 1/10
15052/15052 [==============================] - 10s 679us/step - loss: 0.53
05 - acc: 0.7905 - val_loss: 0.5152 - val_acc: 0.7800
Epoch 2/10
15052/15052 [==============================] - 4s 253us/step - loss: 0.472
7 - acc: 0.7964 - val_loss: 0.4670 - val_acc: 0.7800
Epoch 3/10
15052/15052 [==============================] - 4s 255us/step - loss: 0.425
7 - acc: 0.8129 - val_loss: 0.4399 - val_acc: 0.8018
Epoch 4/10
15052/15052 [==============================] - 4s 254us/step - loss: 0.388
8 - acc: 0.8310 - val_loss: 0.3851 - val_acc: 0.8318
Epoch 5/10
15052/15052 [==============================] - 4s 254us/step - loss: 0.374
6 - acc: 0.8390 - val_loss: 0.3885 - val_acc: 0.8332
Epoch 6/10
15052/15052 [==============================] - 4s 255us/step - loss: 0.369
6 - acc: 0.8378 - val_loss: 0.3696 - val_acc: 0.8491
Epoch 7/10
15052/15052 [==============================] - 4s 254us/step - loss: 0.354
4 - acc: 0.8484 - val_loss: 0.3526 - val_acc: 0.8504
Epoch 8/10
15052/15052 [==============================] - 4s 254us/step - loss: 0.356
2 - acc: 0.8483 - val_loss: 0.3465 - val_acc: 0.8366
Epoch 9/10
15052/15052 [==============================] - 4s 256us/step - loss: 0.348
5 - acc: 0.8505 - val_loss: 0.3915 - val_acc: 0.8257
Epoch 10/10
15052/15052 [==============================] - 4s 256us/step - loss: 0.343
4 - acc: 0.8557 - val_loss: 0.3439 - val_acc: 0.8531
```

In [27]:

```
pd.DataFrame(data=history_callback.history).plot()
```

Out[27]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe36674fe48>
```



Q3.3 Deep model without residual connections

In [0]:

```
from keras.layers import Input, Dense
from keras.models import Model
from keras.layers import Conv2D, MaxPooling2D, Flatten

def run_cnn_deep_nores(layers,dropout,units,X_input, y_input, epochs):
  model = Sequential()
  model.add(Conv2D(256, (3,3), activation='relu', input_shape=(X_train.shape[1:])))
  model.add(Flatten())

  for i in range(0,layers):
    model.add(Dense(units, activation='relu'))
    model.add(Dropout(dropout))

  model.add(Dense(1, activation='sigmoid'))
  model.compile(loss='binary_crossentropy',
            optimizer='adam',
            metrics=['accuracy'])
  history_callback = model.fit(X_input, y_input, batch_size=50, epochs=epochs, verbose=
1, validation_split=0.2)
  return pd.DataFrame(history_callback.history), model
```

In [29]:

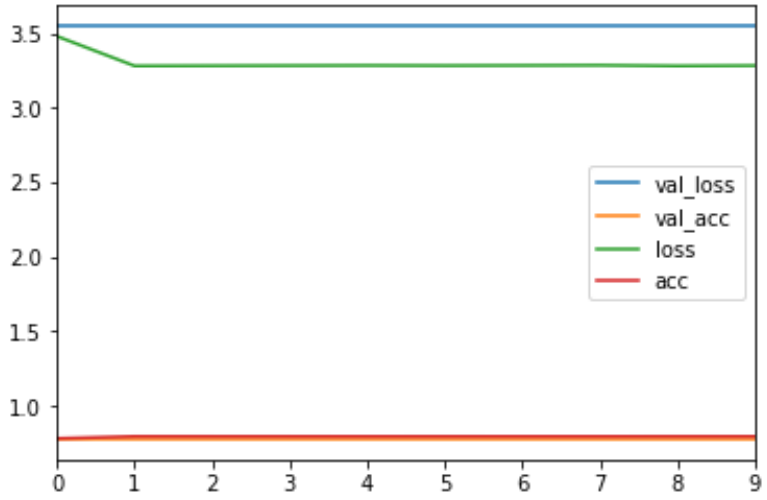
```
results, model_cnn_deep_nores = run_cnn_deep_nores(15,0.2,64, X_train, y_train,10)
print(results)
results.plot()
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backe
nd/tensorflow_backend.py:3445: calling dropout (from tensorflow.python.op
s.nn_ops) with keep_prob is deprecated and will be removed in a future ver
sion.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1
- keep_prob`.
Train on 15052 samples, validate on 3764 samples
Epoch 1/10
15052/15052 [==============================] - 18s 1ms/step - loss: 3.4785
- acc: 0.7828 - val_loss: 3.5456 - val_acc: 0.7800
Epoch 2/10
15052/15052 [==============================] - 15s 1ms/step - loss: 3.2799
- acc: 0.7965 - val_loss: 3.5456 - val_acc: 0.7800
Epoch 3/10
15052/15052 [==============================] - 16s 1ms/step - loss: 3.2810
- acc: 0.7964 - val_loss: 3.5456 - val_acc: 0.7800
Epoch 4/10
15052/15052 [==============================] - 15s 1ms/step - loss: 3.2821
- acc: 0.7964 - val_loss: 3.5456 - val_acc: 0.7800
Epoch 5/10
15052/15052 [==============================] - 16s 1ms/step - loss: 3.2831
- acc: 0.7963 - val_loss: 3.5456 - val_acc: 0.7800
Epoch 6/10
15052/15052 [==============================] - 15s 1ms/step - loss: 3.2821
- acc: 0.7964 - val_loss: 3.5456 - val_acc: 0.7800
Epoch 7/10
15052/15052 [==============================] - 15s 1ms/step - loss: 3.2831
- acc: 0.7963 - val_loss: 3.5456 - val_acc: 0.7800
Epoch 8/10
15052/15052 [==============================] - 16s 1ms/step - loss: 3.2842
- acc: 0.7962 - val_loss: 3.5456 - val_acc: 0.7800
Epoch 9/10
15052/15052 [==============================] - 15s 1ms/step - loss: 3.2799
- acc: 0.7965 - val_loss: 3.5456 - val_acc: 0.7800
Epoch 10/10
15052/15052 [==============================] - 15s 1ms/step - loss: 3.2821
- acc: 0.7964 - val_loss: 3.5456 - val_acc: 0.7800
```

	val_loss	val_acc	loss	acc
0	3.545638	0.780021	3.478505	0.782753
1	3.545638	0.780021	3.279945	0.796505
2	3.545638	0.780021	3.281015	0.796439
3	3.545638	0.780021	3.282075	0.796373
4	3.545638	0.780021	3.283134	0.796306
5	3.545638	0.780021	3.282075	0.796373
6	3.545638	0.780021	3.283122	0.796306
7	3.545638	0.780021	3.284193	0.796240
8	3.545638	0.780021	3.279945	0.796505
9	3.545638	0.780021	3.282075	0.796373

Out[29]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe344da4a58>
```



From the above graphs, it is seen that a deeper network doesn't learn well unless residual connections are introduced.

In [0]: